Spoken Language Understanding (SLU) Module for Movie Domain using NL-SPARQL Data Set

Rodrigo Joni Sestari
number: 179020
rodrigo.sestari@studenti.unitn.it

Abstract

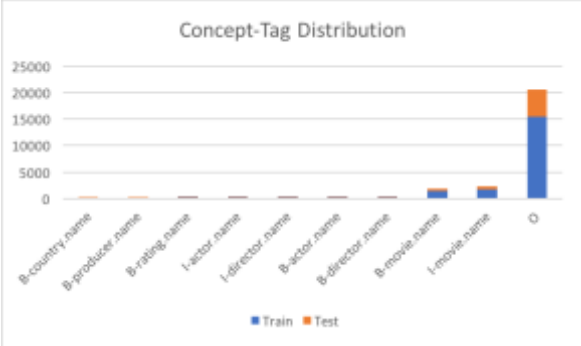
This document has the objective of illustrate the implementation of *Develop Spoken Language Understanding* SLU Module for Movie Domain given a NL-SPARQL Data Set. This document consists in the first project for the course of Language Understanding System of the University of Trento. This project is available on **github.com/rodrigosestari/LUS.git**

1 Introduction

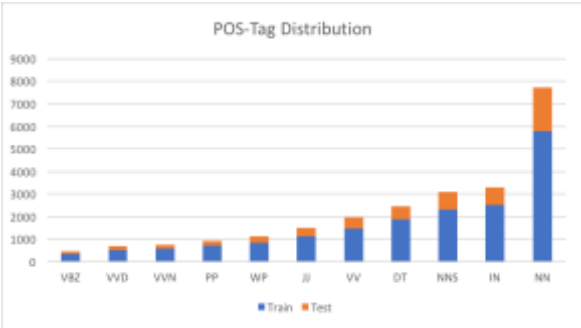
This document describes the steps necessary to implement a SLU module for the Movie Domain. The first part consists in use the Train dataset to create a language model, the second one consist to apply this language model into a test dataset using different smoothing methods, the last part consist to evaluate these results.

2 Data analysis

The datasets available to this project was divided by Train and Test, the first one contains the Concept Tagging format by the tuple <*word, concept*> the concept use **IOB Notation**. The second one contains additional features like Lemma and POS-tags having the structure <*word, POS-tag, count*>. The dataset test contains **1039** tokens and **1084** sentences, while the dataset train contains **1728** tokens and **3338** sentences. In the graph bellow is possible see the Concept-Tag distribution with the 10 more frequents Concept.



The additional dataset allows use Lemmas and POS-Tag. Lemmas can be used to simplify the grammar making it less complex. in this document, the lemma is not used. In the graph bellow is possible the distribution of 10 more frequents POS-Tag.



3 Theory about natural language process

I this chapter introduces some principles about the natural language processing that come used in this document:

- **N-gram**: n-gram is a contiguous sequence of n items from a given sequence of text or speech. In this document, the sequence come from the Dataset.

- **FSA**: *Finite State Automaton,* is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. The FSA can change from one state to another in response to some external inputs; the change from one state to another is called a transition. A FSA is defined by a list of its states, its initial state, and the conditions for each transition. A FSA consists of 4 objects - A set **I** called the input alphabet, of input symbols - A Set **S** of states the automaton can be in; - A designated state **s0** called the initial state; - A designated set of states called the set of **accepting states**.

- **FST**: *Finite State Transducer,* is a FSA whose state transitions are labeled with both input and output symbols. Therefore, a path through the transducer encodes a mapping from an input symbol sequence to an output symbol sequence.

- **WFST**: *Weight Finite State Transducer,* A weighted transducer puts weights on transitions in addition to the input and output symbols. Weights may encode probabilities, durations, penalties, or any other quantity that accumulates along paths to compute the overall weight of mapping an input sequence to an output sequence. Weighted transducers are thus a natural choice to represent the probabilistic finite-state models prevalent in speech processing.

- **LM**: *Language Model* compute the probability of a sentence or sequence of words: $P(W) = P(w_1, w_2, w_3 \ldots w_n)$

- **OOV Rate:** % of word tokens in test data that are not contained in the lexicon

- **Smoothing**: Assume that no n-gram of known words has 0 probability and redistribute probability mass from seen to unseen events, this is known as smoothing. In this document, the methods which are used are Witten-Bell, Absolute, Pre-smoothed, Katz Unsmoothed and Kneser-Ney.

4 Tools

To execute the tasks about this document was necessary the following tools:

- **OpenFST**: an open-source library for weighted finite-state transducers WFSTs. *OpenFst* consists of a C++ template library with efficient WFST representations and over twenty-five operations for constructing, combining, optimizing, and searching them.

- **OpenGRM**: is used for making and modifying n-gram language models encoded as weighted finite-state transducers (FSTs). It makes use of functionality in the *OpenFst* library to create, access and manipulate n-gram models. Operations for counting, smoothing, pruning, applying, and evaluating models as well as support for distributed computation are among those provided.

- **Conlleval**: A Perl script that allow evaluate the SLU result.

5 Implementation

The implementation consists to execute some script to train and test the LM, thesis script uses external tools like *OpenFST* and *OpenGRM*.

5.1 Scripts

Some scripts are implemented to elaborate chain of input/output files. The scripts are:

- **start.sh**: Used to start the Train/Test process, contains 3 parameters, smoothing, grammar and threshold. The parameter smoothing represents the smoothing method that can be (1=*Absolute*, 2=*Katz*, 3=*Kneser Ney*, 4=*Pre-smoothed*, 5=*unsmoothed*, 6=*Witten bell*). The grammar parameter represents the n-gram order the interval is between 1-5, the last parameter represents the Cut-off threshold, with 0 without cut-off. if threshold is negative, that script run all possible combination of method n-grammar and threshold.
- **trainTest.sh**: Script used to train and test the ML.
- **transducer_UNK.sh**: Create the Transducer files for known tokens.
- **transducer.sh**: Create the Transducer files for Unknown tokens.

5.2 Train

Initially was created two files, *POS.counts* that represents $C(t_i)$ that contains the tuple <*tag, count*>, the second one called *TOK_POS.counts* represent $C(t_i, w_i)$ is represented by the triple <*word, tag, count*> having these two files, the next step was create the file *TOK_POS.probs* this file represents the probability of see the current word given the current tag $P(w_i|t_i)$

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

If $C(t_i, w_i)$ <= threshold Cut-off then the probability is not put into *TOK_POS.probs*

The last file to be create is the Lexicon file, its contains the vocabulary used in the training, The Lexicon file was create using the file *TOK_POS.counts*, in the head file was put *"<epsilon> 0"* and at the end of file was put *"<unk> #Number all tokens"*.

The next step was created the FST Transducer file called *transducerTOK_POS.txt*, this file was create using the file *TOK_POS.probs*, during this process the probabilities are changed by weights, using this formula: $-\log(P(w_i|t_i))$. Instead the probability of unknown words is given by:

$$P(< unk > | tag) = \frac{1}{|Tags|}$$

having the file *transducerTOK_POS.txt* that is represents <*initial step, final step, word, tag, cost*> the next step was generated the WFST. The WFST is create using the *farcompilestrings*, the result became the input to *OpenGRM* tool, the first command used is *ngramcount* that create the probability of a tag given the n-1 predecessors. In this document was used 1-5 n-grams, the output is used to create the language model, this document use 5 smoothing methods, it allows create different LM based on the combination between n-gram, smoothing method and cut-off.

5.3 Test

There are some procedures to be executed in the test phase, initially was created three files, the files *tok_actual.txt* and *sentences.txt* will be used together with result test output to be evaluate.
Instead the file *sentences_line.txt* contains the sentences by line, for each line is used command *farcompilestrings* to create the FAR file, it is spitted into a FST, each fst is compose with LM and Train FST, this process is made for each line and the result is salved into *tok_predicted.txt* file
The files *sentences.txt tok_actual.txt* and *tok_predicted.txt* are merged creating the triple is <*word, actual-tag, predict-tag*>. The final step is evaluated it with *conlleval.pl*, it allows to produce the table with the quality of the models.

6 Results

The idea is measure the output given by the SLU model.

6.1 Evaluation Methods

- **accuracy**: Accuracy refers to the closeness of a measured value to a standard or known value.

$$Accuracy = \frac{Num.of\ Correct\ Decisions}{Total\ Num.of\ Instances}$$

- **precision**: Precision refers to the closeness of two or more measurements to each other or also called positive predictive value, is the fraction of retrieved instances that are relevant.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- **recall**: recall is the fraction of relevant instances that are retrieved in another way is a measure of how many truly relevant results are returned.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- **FB1**: is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score

$$F_\beta = \frac{(1 + \beta^2) * Precision * Recall}{(\beta^2 * Precisioin + Recall)}$$

$$F_1 = \frac{2 * Precision * Recall}{(Precisioin + Recall)}$$

6.2 Evaluation

In the tables above is possible see the result of evaluation, the tables are divided by smoothing method, the column n-gram represent the n-gram order considered, the next 3 columns represent the evaluation methods; accuracy, precision, recall and F-measure.

Absolute method

n-gram	accuracy	precision	recall	FB1
1	88.84%	55.51%	60.04%	57.68
2	92.69%	78.51%	74.34%	76.37
3	92.65%	76.67%	74.70%	75.67
4	92.85%	77.16%	75.25%	76.19
5	**92.88%**	76.97%	75.34%	76.15

Katz method

n-gram	accuracy	precision	recall	FB1
1	88.84%	55.51%	60.04%	57.68
2	92.62%	78.03%	73.88%	75.89
3	92.09%	75.60%	72.69%	74.11
4	91.88%	72.57%	73.97%	73.26
5	88.16%	56.95%	71.31%	63.33

Kneser Ney method

n-gram	accuracy	precision	recall	FB1
1	88.84%	55.51%	60.04%	57.68
2	92.68%	78.41%	74.24%	76.27
3	92.64%	76.67%	74.70%	75.67
4	92.78%	76.87%	75.53%	76.19
5	**92.90%**	76.79%	75.53%	76.16

Pre-smoothed method

n-gram	accuracy	precision	recall	FB1
1	88.84%	55.51%	60.04%	57.68
2	**92.65%**	78.41%	74.24%	76.27
3	90.74%	64.81%	71.40%	67.95
4	89.97%	62.28%	72.50%	67.01
5	90.00%	62.36%	72.14%	66.89

Unsmoothed method

n-gram	accuracy	precision	recall	FB1
1	88.84%	55.51%	60.04%	57.68
2	**92.54%**	78.09%	73.88%	75.93
3	92.44%	75.91%	73.91%	74.90
4	92.35%	76.07%	73.94%	74.99
5	92.38%	76.16%	73.69%	74.91s

Witten-bell method

n-gram	accuracy	precision	recall	FB1
1	88.84%	55.51%	60.04%	57.68
2	92.68%	78.51%	74.34%	76.37
3	92.62%	76.58%	74.61%	75.58
4	92.86%	77.11%	75.34%	76.22
5	**92.90%**	77.03%	75.62%	76.32

7 Conclusion

After train test and evaluate all methods of smoothing is possible get some conclusions.

Is evident that the accuracy is equal for all smoothing methods with unigram, the motive can be the fact that unigram use the word frequency so the probability becomes the same for al methods. The method with the worse accuracy median was

Pre-smoothing, instead Witeen-bell with 5 n-grams was the best with 92.90% of accuracy. Is important observe that the difference performance between the methods with the same n-gram order is very small.

References

John Wiley and Sons. New York, 2011. *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*

Maccartney, B. *(2005).* Stanford. from *https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf*

Mohri et al. (1996) *FSM Toolkit Exercises*

Riccardi (2017). UniTN. Retrieved 10 April, 2017, from http://disi.unitn.it/~riccardi/